The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

EEEE1039

Applied Electrical and Electronic Engineering:

Construction Project

Online Coursework – Spring

Algorithm Development for

Facial Recognition in Digital Video

using CodeBlocks integrated with OpenCV

Department of Electrical and Electronic Engineering

University of Nottingham Ningbo China

Student's Name: Jingxiong YU

Student's ID: 20123738

Group: 7

Tutors' Names: Dr. Jing WANG, Dr. Chunyang GU,

Dr. Sherif WELSEN, Dr. Fei CHEN

Submission Date: 25th April 2020

Abstract

In this report, the process of the development of a facial recognition algorithm for digital videos using CodeBlocks integrated with OpenCV was presented. A brief introduction of facial recognition and methods to realize it were covered in this report. The main algorithm integrated in this facial recognition algorithm developed in this report was template matching, and the pros and cons as well as its application restrictions and factors that would affect the recognition accuracy were studied and discussed in detail. The algorithm was successfully developed in the end which fully realized the objectives of this project.

Introduction

Nowadays, artificial intelligence had become a household phrase. As a vital branch of the artificial intelligence, computer vision was said to be the next frontier domain in the future. In order to make a machine to understand the real world, the sight must be given to them. Facial recognition, as one specific type of computer vision, had already got wide applications in our daily life as great as automatic driverless drones or as simple as unlocking an iPhone. Also, in future lab sessions, the computer vision algorithm was to be integrated onto the line follower robot to enable the robot to interact with the signals obtained from the real world.

In this report, the algorithm for facial recognition in digital video using CodeBlocks integrated with OpenCV was developed. CascadeClassifier, an embedded class of OpenCV was integrated and the MatchTemplate algorithm was used in this project. The objective of this algorithm was to detect faces in a digital video and identify the researcher's face from detected faces. And the algorithm was successfully developed.

The hardware used in this project for video recording was the smart phone iPhone 11 Pro, and for image processing, the personal laptop SurfaceBook 2 installed with CodeBlocks integrated with OpenCV was used.

The software used in this project was CodeBlocks integrated with OpenCV. Only the libraries from CodeBlocks and OpenCV were used.

Relevant Background

As shown in Figure 1, OpenCV (Open Source Computer Vision Library) was an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. (https://opencv.org/about/, no date)

Figure 1. OpenCV

Facial recognition was a technology of identifying/verifying a person from a digital image or video frame from a video source, by comparing selected facial features from given image with faces within a database. Facial recognition was also referred as Biometric Artificial Intelligence. There were challenges for facial recognition due to great variability including head rotation, tilt, lighting intensity, lighting angle, facial expression, scale (distance from the camera), aging, environmental noises and so on. And also, facial recognition had got wide applications including mobile platforms, security services, robotics, human-computer interaction, image searching by person and so on. Facial recognition had got many advantages including contactless, non-invasive, cooperation unnecessary and also it was able to recognize from crowd. However, facial recognition had got lower accuracy than iris or fingerprint recognition.

There were many techniques that could be integrated to realize facial recognition including feature-based techniques, template matching technique, 3-Dimensional recognition, skin texture analysis as well as using thermal cameras. These techniques had got different pros and cons. The technique used in this report was template matching.

General steps of template matching technique were as follows:

1. Compress the face data from a gallery of face images.
2. Save the data useful for facial recognition – a probe image.
3. Compare with database.

Face detection and facial recognition sounded similar, but they were quite different owing to different objectives. Face detection was mainly focusing on detecting faces from a given image or a video frame while facial recognition mainly focusing on identifying the specific face from the detected faces. Therefore, in order to realize the given objective which was to detect faces from a digital video and identify the specific face of the researcher from the detected faces, this project was divided into two processes: face detection and facial recognition.

Methods

The most important two methods used in this project were CascadeClassifier and MatchTemplate. These were two embedded libraries of OpenCV. The CascadeClassifier was used in face detection and the MatchTemplate was used in facial recognition to identify the specific face from detected faces in the face detection process.

CascadeClassifier

Haar feature-based CascadeClassifier was used to detect faces in a digital video in this project which was an effective object detection method proposed by Paul Viola and Michael Jones in their paper, “Rapid Object Detection using a Boosted Cascade of Simple Features” in 2001. It was a machine learning based approach where a cascade function was trained from a lot of positive and negative images. And it was then used to detect objects in other images.

Initially, the algorithm needed a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then the features needed to be extracted from it. For this, Haar features shown in Figure 2 were used. They were just like convolutional kernel. Each feature was a single value obtained by subtracting sum of the pixels under the white rectangle from sum of pixels under the black rectangle.
(https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html, no date)

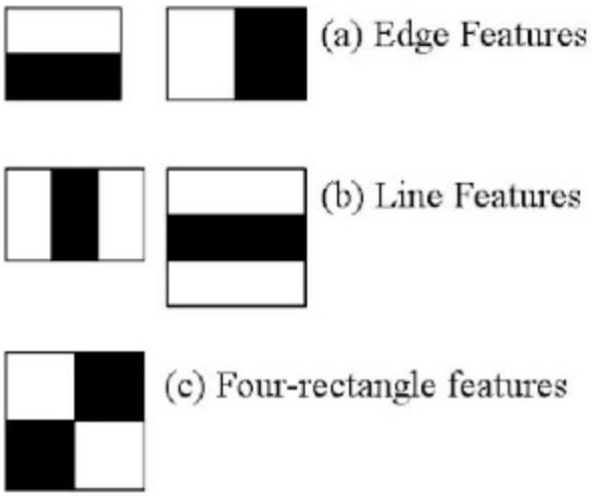

Figure 2. Haar features

However, if lots of Haar features were to be calculated, a lot of computational resources were required to process even a very small image with few pixels. Therefore, a method similar to DP (dynamic programming) called Integral Image was used to boost the calculation.

But still, even with the Integral Image method used, it was still a little inefficient and time consuming to apply all the Haar features to the image to detect objects (i.e. faces). Hence, a concept of Cascade of Classifiers was then introduced. In brief, the features were grouped into different stages with different weights and applied one after another. If the previous stage failed, then the next stage would not be applied. By doing so, the programme was detecting faces in the areas where faces were more likely to be detected which saved a lot of time.

Both a training method and pretrained models were provided by OpenCV, and the pretrained models were used in this project. The piece of code shown in Figure 3 was used to load the pretrained model and call the DetectMultiScale function.

```
string face_cascade_name = "haarcascade_frontalface_alt.xml";
CascadeClassifier face_cascade;
//load cascades
if(!face_cascade.load(face_cascade_name)){cout<<"Error! Face cascade not loaded.\n";return -1;}
cout<<"Face cascade successfully loaded.\n";
//detect faces
//parameters: (image,objects,scaleFactor,minNeighbors,flags,minSize,maxSize)
face_cascade.detectMultiScale(frame_gray,faces,1.05,6,0,Size(30,30),Size());
```

Figure 3. Relevant code for CascadeClassifier

The pretrained model used in this project to detect frontal faces in digital video frames was “haarcascade_frontalface_alt.xml”. The model was loaded using CascadeClassifier embedded class function “face_cascade.load (string xml_file_path)”. The “detectMultiScale()” function was used to realize multiscale detection, it detected objects from the maximum size to the minimum size step by step. Once the object was detected, a set of rectangle values was returned in “vector<Rect>&” type. There were several parameters to be set to call the function:

- Image: the imported frame to be detected.
- Objects: a set of rectangle (including the detected object) values in “vector<Rect>&” type.
- ScaleFactor: the zoom ratio of the windows’ sizes between two detections. It was set to 1.05.
- MinNeighbors: the number of rectangles retained adjacent to the candidate rectangle. It was set to 6.
- Flags: the operating mode, unnecessary in the new version of CascadeClassifier. It was set to 0.
- MinSize: the minimum detection size, objects smaller than it were neglected. It was set to 30*30 (i.e. Size(30,30)).
- MaxSize: the maximum detection size, objects larger than it were neglected. It was not set (i.e. Size()).

By using CascadeClassifier, the rectangles containing faces were obtained for further process.

MatchTemplate

After the rectangles containing detected faces were obtained, the template matching algorithm was then integrated to identify the specific face of researcher from detected faces.

Template matching was a technique for finding areas of an image that match (were similar) to a template image (patch). (https://www.docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/template_matching/template_matching.html, no date)

To apply template matching algorithm, two primary components were needed: source images which in this case were the frames captured from the video source and a template image. And the objective of this template matching algorithm was to detect the highest matching area. In order to identify the matching area, the template image had to be compared against the source frame by sliding it. By sliding, it was meant moving the patch one pixel at a time (left to right, up to down). A matric was calculated at each location representing the similarity between the patch and the particular area of the source frame.

There were 6 methods of calculating the matric embedded in OpenCV embedded function matchTemplate. The one used in this project was "TM_CCOEFF_NORMED" as shown in Figure 4.

method=CV_TM_CCOEFF_NORMED

$$R(x,y) = \frac{\sum_{x',y'}(T'(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2}}$$

Figure 4. Template matching method

Instead of using template matching to realize both face detection and facial recognition at the same time, the template matching was applied after the faces were detected by CascadeClassifier to improve face detection accuracy and reduce the difficulty to set the threshold value for template matching algorithm.

As shown in Figure 5, the template matching algorithm was applied onto the detect faces to obtain similarities between the face template and the detected faces. A threshold value of similarity was then set to 15% to distinguish the specific face of the researcher from other detected faces.

```
for(vector<Rect>::const_iterator r=faces.begin();r!=faces.end();r++,i++)
{
    //link parameters
    Rect face_rect;
    Mat frameROI;
    face_rect.x=r->x;
    face_rect.y=r->y;
    face_rect.width=r->width;
    face_rect.height=r->height;
    frameROI=frame_gray(face_rect);

    //text highlight
    //matchTemplate algorithm integrated
    Mat result;//create a new array to store results
    matchTemplate(frameROI,temp,result,TM_CCOEFF_NORMED);//best match==1;less the value worse the match
    //obtain locations
    double minVal=-1;
    double maxVal;
    Point minLoc;
    Point maxLoc;
    minMaxLoc(result,&minVal,&maxVal,&minLoc,&maxLoc,Mat());
    //calculate similarity
    double similarity=maxVal*100;
    //highlight zone with maximum similarity (text box)
    if(similarity>15)//less the maxVal worse the match
        {putText(frame,text_1,origin_1,font_face,font_scale,colors[0],thickness,8,0);color=colors[0];}
    else
        {putText(frame,text_2,origin_2,font_face,font_scale,colors[1],thickness,8,0);color=colors[1];}
    printf("Face[%d] Similarity: %.2f%\n",i,similarity);
}
```

Figure 5. Relevant code for MatchTemplate

The matrix “result” was used to store template matching calculation results. The template matching operation was performed via “matchTemplate(source_frame,template_image,result,match_method);”. The minimum and maximum values in the result matrix “result” were localized using minMaxLoc. Several parameters needed to be set in order to call the minMaxLoc function:

- Result: the source array (i.e. the matrix storing template matching results).
- &minVal and &maxVal: variables to save minimum and maximum values in result.
- &minLoc and &maxLoc: the point locations of the minimum and maximum values in the array.
- Mat(): operational mask.

For the method used in this project, higher values represented better matches. Therefore, the obtained maxVal was then used to calculate the similarities between the face template and the detected faces.

VideoCapture and VideoWriter

The VideoCapture was used to read frames from the local digital video file, and the VideoWriter was used to write the processed frames into a newly generated video file.

```
//import the video file
VideoCapture capture("./Original_Video_360p.avi");
if(!capture.isOpened())//check if the video file was loaded
{
    cout<<"Error! Video File not loaded.\n";
    return -1;
}cout<<"Video File successfully imported.\n";
//obtain video size for recorder
Size video_size(capture.get(CAP_PROP_FRAME_WIDTH),capture.get(CAP_PROP_FRAME_HEIGHT));
//run facial rec on each frame of the video file
while(capture.read(frame)){}
//release all objects and resources
capture.release();
```

Figure 6. Relevant code for VideoCapture

As shown in Figure 6, the local digital video file was imported via VideoCapture. The video frame size was obtained using embedded function “capture.get()” for VideoWriter. While running facial recognition, the VideoCapture was used to read a single frame from the video file to be processed via embedded function “capture.read()” at a time. After the VideoCapture was finished calling, the objects and resources were released using embedded function “capture.release()”.

```
//initialize video recorder
VideoWriter vidRec;
vidRec.open("Processed_Video.avi",VideoWriter::fourcc('M','J','P','G'),fps,video_size);
if(!vidRec.isOpened())//check if the video recorder was ready
{
    cout<<"Error! Video Recorder not ready.\n";
    return -1;
}cout<<"Video Recorder ready.\n";
while(capture.read(frame))
{
    //process the frame
    vidRec.write(frame);
}
//release all objects and resources
vidRec.release();
```

Figure 7. Relevant code for VideoWriter

As shown in Figure 7, the VideoWriter was used to generate the processed output video via embedded function “vidRec.open()”. Several parameters needed to be set to call the function:

- Output_video_path: path and name with extension for processed video.
- Encode_method: the method to encode the processed video. It was set to “MJPG”. For general “.avi” video, it could be set to 0 for OpenCV version 4.x.
- Fps: frames per second. It was set to 30 fps manually.
- Video_size: video frame size obtained from the original video source via VideoCapture. It was set automatically.

The frames were written to the processed video using embedded function "vidRec.write(&Mat)". And also, after the VideoWriter was finished using, the embedded function "vidRec.release()" was then called to release all objects and resources. This last step was important to ensure the processed video to be successfully generated.

FFmpeg

FFmpeg was a complete, cross-platform programme to record, convert and stream audio and video which was so important that the VideoCapture and the VideoWriter were unable to process local video files or generate processed videos without it.

Imread and Imshow

As shown in Figure 8, imread was used to import the images, and in this case the template. The parameter after the file path was set to 0 to import a grayscale image using imread directly without calling cvtColor embedded function.

```
1  //import a gray template
2  Mat temp = imread("./Template_001.jpg",0);
3  if(temp.empty())//check if the template was loaded
4  {
5      cout<<"Error! Template not loaded.\n";
6      return -1;
7  }cout<<"Template successfully imported.\n";
```

Figure 8. Relevant code for imread

Imshow was used to display images (or frames). And in this project, it was used to display the processed frames frame by frame as shown in Figure 9.

```
1  //display the processed frame
2      imshow("Facial Rec Running...",frame);
3      waitKey(1000/fps);
```

Figure 9. Relevant code for imshow

The waitKey(time_in_ms) embedded function needed to be called to display frames or images in named windows, otherwise, the frames or images were unable to be displayed and the programme was to break down.

Image Processing

The raw frames read from the video or the raw template image could not be used in the CascadeClassifier or the MatchTemplate algorithm directly, however, they needed to be processed in several processes using different image processing methods.

The image processing methods used in this project were shown in Figure 10.

```
1  //template processing
2  resize(temp,temp,Size(100,100));
3  //convert the frame to gray to apply Haar-like algorithm
4  cvtColor(frame,frame_gray,COLOR_BGR2GRAY);
5  //equalizeHist the resized frame
6  equalizeHist(frame_gray,frame_gray);//enhance the frame
```

Figure 10. Image processing methods

The imported template image was resized to be used by matchTemplate and increase matching accuracy via embedded function “resize(inputArray,outputArray,outputSize);”.

There were more than 150 colour-space conversion methods available in OpenCV, and one of the most widely used methods was applied in this project to process both the template and the frames obtained from the video source: RGB to Gray. These colour conversion methods were mainly based on filtering or thresholding which were also important methods in image processing. Here, the embedded function “cvtColor(inputArray,outputArray,convert_methods);” was called to realize the colour conversion from RGB to Grayscale when the “convert_methods” parameter was set to “COLOR_BGR2GRAY”.

After the images were converted to grayscale, the “equalizeHist(inputArray,outputArray);” function was called to enhance the converted images. Histogram equalization was used to adjust the grayscale distribution of the images to make the distribution on the grayscale from 0 to 255 more balanced, hence the contrast of the image was improved and the goal of improving the subjective visual effect of the image was then achieved. Low contrast images were suitable for histogram equalization to enhance image details. After equalization, the images were approximately uniformly distributed, and the dynamic ranges of the equalized images were expended. However, this was to expand the quantization interval while the quantification level decreasing. Thus, after processing, the original pixels with different grayscale might become the same, forming the same grayscale area.

Highlight Methods

Two different highlight methods were used in this project including “rectangle()” and “putText()”. The “rectangle()” was used to draw rectangles containing detected faces, and the “putText()” was used to draw text boxes to indicate the facial recognition results (i.e. whether the detected faces matched the template).

```
rectangle(
    frame,
    Point(cvRound((r->x)),cvRound((r->y))),
    Point(cvRound((r->x+r->width-1)),cvRound((r->y+r->height-1))),
    color,3,8,0);
```

Figure 11. Relevant code for rectangle

As shown in Figure 11, the rectangle function was called to draw rectangles to highlight the detected faces using the rectangles’ coordinate and size values obtained via CascadeClassifier. Several parameters needed to be set in order to call the rectangle function:

- InputOutputArray: the image or frame to draw the rectangles onto.
- Point_1: vertex of the rectangle.
- Point_2: vertex of the rectangle opposite to Point_1.
- Colour: rectangles’ colour or brightness (grayscale image).
- Thickness: thickness of lines that made up the rectangle. It was set to 3.
- LineType: different types of the line. It was set to 8 to draw 8-connected lines.
- Shift: the number of fractional bits in the point coordinates. It was set to 0.

```
//set text box parameters
string text_noFace = "No Face Detected!";
int font_face_noFace = FONT_HERSHEY_COMPLEX;
double font_scale_noFace = 2;
int thickness_noFace = 2;
int baseline_noFace;
//obtain the size of the text box
Size text_noFace_size = getTextSize(text_noFace,font_face_noFace,font_scale_noFace,thickness_noFace,&baseline_noFace);
//draw text
Point origin_noFace;
origin_noFace.x=cvRound(frame.cols*0.5-text_noFace_size.width*0.5);
origin_noFace.y=cvRound(frame.rows*0.5+text_noFace_size.height*0.5);
putText(frame,text_noFace,origin_noFace,font_face_noFace,font_scale_noFace,colors[2],thickness_noFace,8,0);
```

Figure 12. Relevant code for putText

The putText function was used to draw text boxes onto the frames to display the name or messages to indicate the facial recognition results as shown in Figure 12. Several parameters needed to be set including the text contents, font face, font scale and thickness. And the origin needed to be set to locate the text box, also the line type was set to 8. The last parameter was useless and unimportant therefore it was set to 0.

Different texts were put into the text boxes to indicate different facial recognition results. If the detected face matched the template, the name of the researcher was displayed above the rectangle. Else if the detected face did not match the template, an "UNKNOWN" string was displayed above the rectangle. Else if there was no face detected, the "No Face Detected!" message was displayed at the centre of the frame.

Also, different colours were used to indicate different facial recognition results. If the specific face similar to the template was detected, the yellow colour was used to highlight. Else if the other faces were detected, the white colour was used to display the detection results. Else if there was no face detected, the red colour was used to display a message "No Face Detected!".

Mat Class

The Mat class (short for Matrix) was an encapsulated class introduced by OpenCV for processing images which was widely used in this project and applied by functions and algorithms, also it has its embedded class functions to be called.

General Methods

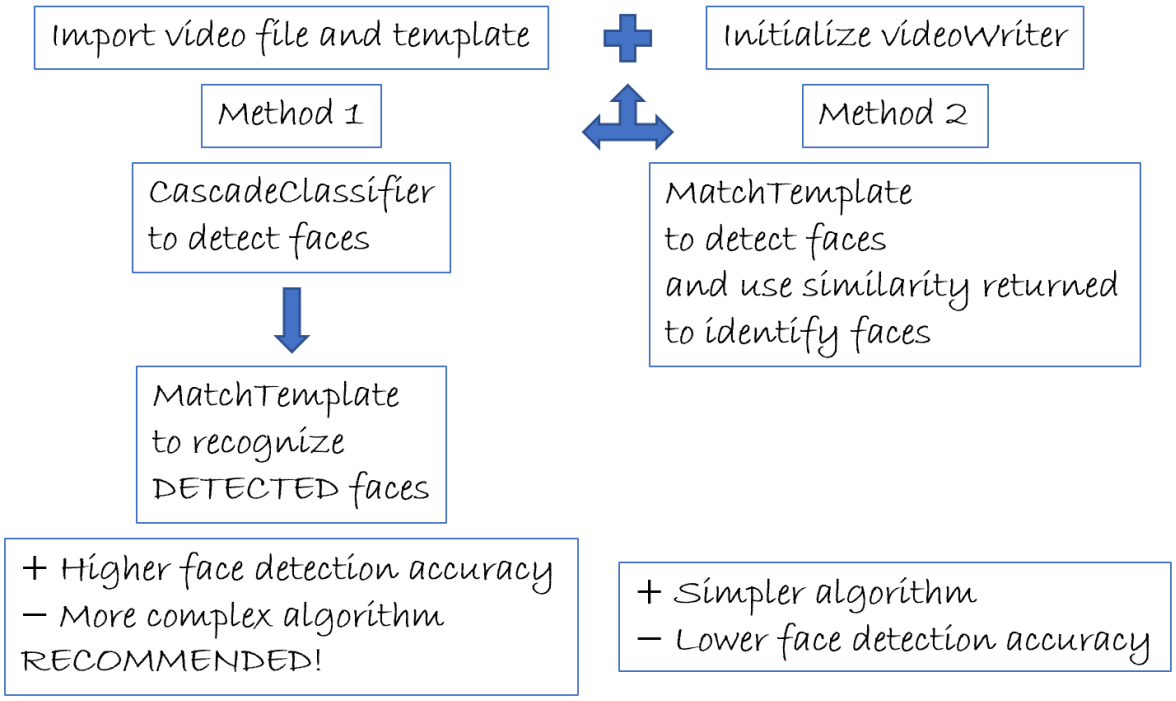

Figure 13. General methods

Results and Discussions

General Method Selection

Generally, there were two possible general methods using template matching algorithm as shown above in Figure 13. The first methods which was used in this project was to detect faces using CascadeClassifier and recognize and identify detected faces using template matching algorithm. And the second was to detect faces directly using template matching algorithm and identify detected faces using the similarity returned. These two approaches had both got pros and cons, the first method used in this project improved the accuracy of the face detection sacrificing the conciseness of the algorithm.

If only the template matching algorithm was applied, the face detection and identification were both depended on threshold values which lowers the accuracy of face detection while largely increasing the difficulty to set the threshold values. Therefore, the first method was better and was selected in this project.

VideoIO

The videoIO (i.e. video Input and Output) was the very foundation of this project. However, unexpected errors occurred when importing the local video files. The VideoCapture was able to import video source via computer frontal camera but not the local video file. The problem was then found that the FFmpeg was not downloaded successfully during the configuration process due to certain network restrictions. After the FFmpeg was downloaded separately and applied in the configuration process, the VideoCapture was able to import local video files.

However, with the FFmpeg fixed, the video output process was still facing unexpected errors that the newly generated processed video file was damaged which was unable to be loaded and played with an unreasonable size of 6KB. Firstly, it was considered that the problem was caused by invalid encoders, however, even after various of encoders and encoding methods were attempted, the problem was not fixed. Thus, in order to solve this vital problem, many approaches were attempted. Eventually, it was discovered that the difference of the video frame sizes between raw video frames and the processed video frames caused the problem. The problem was then solved by applying embedded class function of VideoCapture to obtain the original video frame size which was used later by VideoWriter function.

When running the integrated programme, the processing speed was so slow that it was considered that there might be enhancement approaches existed. The enhancement was done by lowering the resolution of the imported video from 1280p to 360p. Processing frames or

videos with high resolution required a lot of computational resources which greatly slowed down the processing speed. Also, the decrease in input video resolution had got slight influence on the facial recognition accuracy or quality. Meanwhile, if the videos with high resolution were to be processed, further re-deployment of the algorithm was needed.

Image Processing

As mentioned earlier in this report, the raw video frames or raw template image could not be applied in functions or algorithm directly, because it was difficult to process images with RGB colours and also the Haar features were to be applied onto grayscale images. Therefore, image processing process was necessary. Here, two image processing methods were discussed: colour conversion from RGB to Grayscale and Histogram equalization.

```
1   cvtColor(in_image,out_image,COLOR_BGR2GRAY);
2   //or convert during image importing
3   in_image=imread("Template_001.jpg",0);
```

Figure 14. Relevant code for colour conversion: RGB to Grayscale

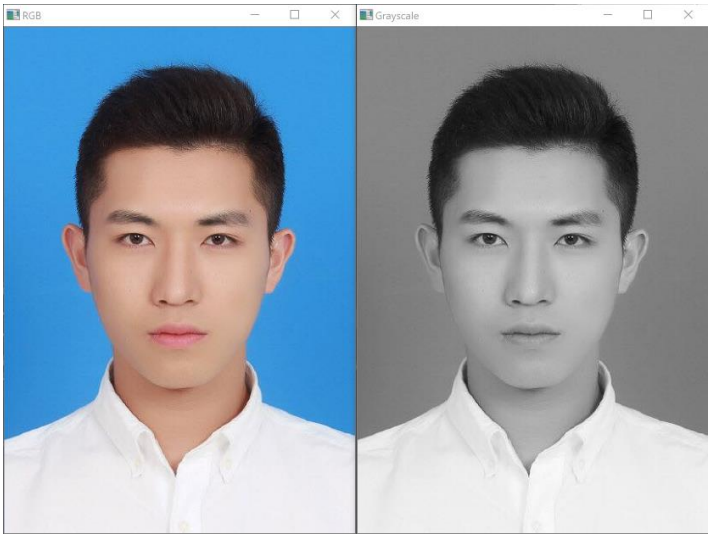

Figure 15. Colour conversion result

Either one of the two pieces of code shown in Figure 14 could be used to realize colour conversion from RGB to Grayscale. The colour conversion result was shown in Figure 15. Since the CascadeClassifier was Haar feature based and the Histogram equalization was used to balance the grayscale distribution, the images needed to be converted to grayscale to apply pretrained models and the Histogram equalization algorithm. Also, largely reduced computational resources after the colour conversion resulted in boosting the facial recognition process.

Another method Histogram Equalization was applied using the piece of code shown in Figure 16, and the result was shown in Figure 17.

```
1    equalizeHist(temp,outImg);
```

Figure 16. Relevant code for Histogram equalization

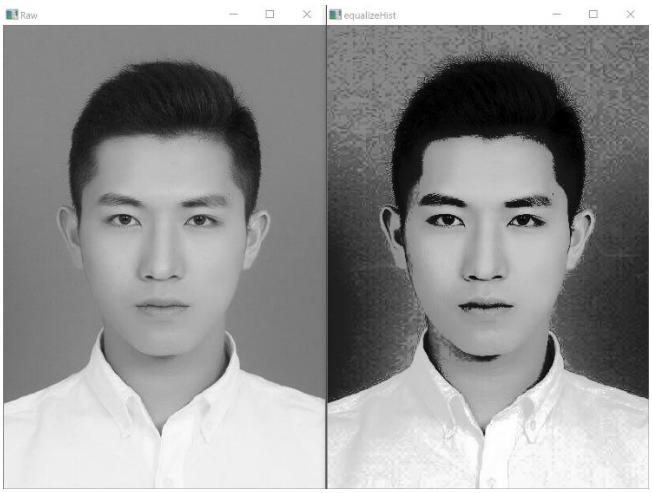

Figure 17. Histogram equalization result

As shown in Figure 17, the grayscale raw image was enhanced using Histogram equalization. Comparing the processed image with the raw image, the contrast was increased which made it easier for the programme to identify different features. Hence, the processing speed was largely increased. It was noted that the Histogram equalization could only be applied onto grayscale images, otherwise, errors occurred and the programme broke down when the Histogram equalization algorithm was applied onto RGB coloured images. The importance of the colour conversion from RGB to grayscale was strongly indicated.

Also, resize function was called several times in this project to adjust the size of the images to realize certain purposes. The sizes of both the template and the video frame had influence on processing speed and accuracy. Images (or frames) were stored as arrays using class Mat, therefore, larger images had got more pixels resulted in more computational resources to process on each pixel which slowed the programme down. The images could be resized to smaller images to boost the processing speed, however, by doing so, the accuracy was lowered owing to pixel loss during the resizing. In order to maintain the accuracy at a high level, the originally integrated “scaleFactor” used to zoom the frames to increase the processing speed was eventually commented and disabled. Though the processing speed was lowered resulted in frame lags while running the programme, the generated processed video was unaffected with a high facial recognition accuracy. And if the size of the template was too large or too small comparing with the detected faces, the template matching algorithm was unable to function properly where the resize function was needed.

Face Detection

The objective of face detection was to detect all faces in images or video frames.

Figure 18. Face detection result – I

Figure 19. Face detection result – II

Two different pretrained models were applied onto the same image via CascadeClassifier and different results were generated as shown in Figure 18 and Figure 19. The pretrained model applied resulted in generating the first result was “haarcascade_frontalface_alt.xml” and the pretrained model “haarcascade_frontalface_default.xml” was applied to generate the second result.

It could be seen from the results that both pretrained models had pros and cons. The first model had got a higher threshold value than the second, therefore, there were two faces with uncommon expressions not detected when others perfectly detected and highlighted. Also, there was an area on the floor wrongly detected as a face, but in general the detection accuracy was still at a high level using the first pretrained model. A lower threshold value in the second pretrained models resulted in two significant differences in the detection result comparing to the first. All the faces with or without unusual expressions were detected, however, more areas where no face existed were highlighted as faces which significantly lowered the detection accuracy. In order to maintain the detection accuracy at a relatively high level, the first pretrained model “haarcascade_frontalface_default.xml” was applied in this project.

Further discussion regarding the overall project detection results and accuracy was covered in the Overall Project Results discussion part together with facial recognition results.

Facial Recognition

The objective of facial recognition was to distinguish and identify the specific face matching the template from detected faces. Several questions were discussed as follows:

- **For the template matching algorithm used, what were the application restrictions?**

  There were application restrictions for the template matching algorithm used in this project. In order to match successfully, the algorithm needed to be applied under certain circumstances. And its accuracy was easily affected by various factors. The template matching algorithm was the most original and the most fundamental object detection and match method studying in the areas where the specific objects were located. Its application restrictions lied in that the template was only able to move parallelly to be matched. If either the template or the frame to be matched was rotated or zoomed, the template matching algorithm would fail.

  Here, as shown in Figure 20, the template was rotated 90 degrees and was then applied in the same facial recognition programme. The result generated was shown in Figure 21.

Figure 20. Rotated template

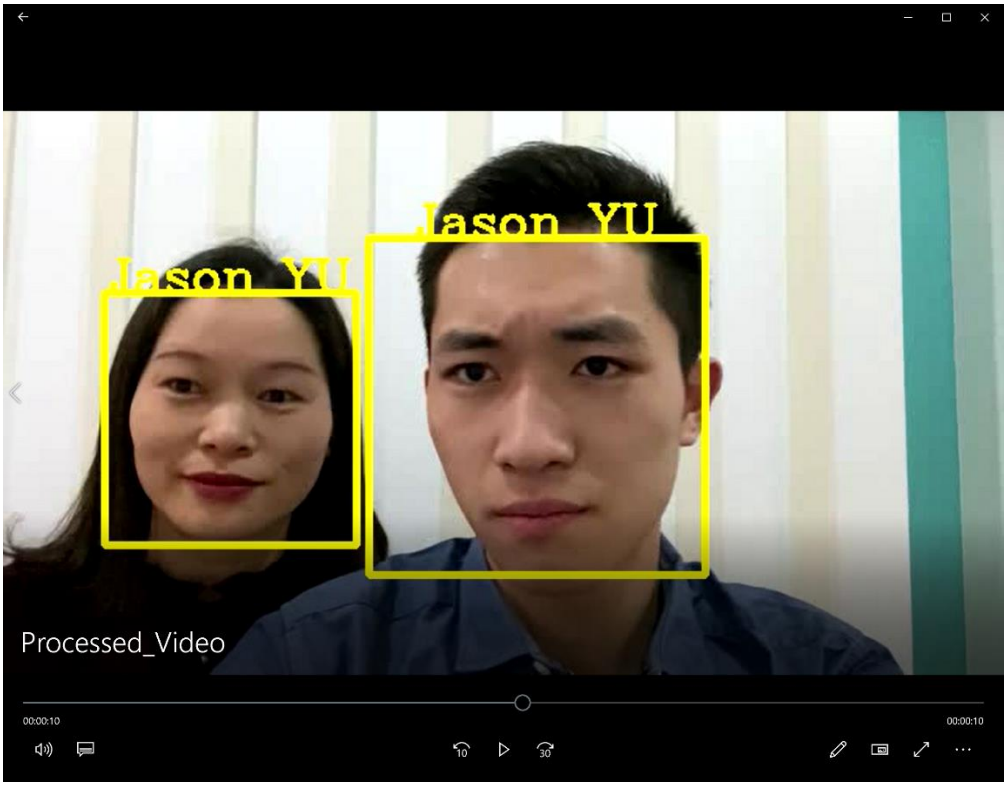

Figure 21. Facial recognition result using rotated template

As shown in the Figure 21 above, the fully operational programme was failed in distinguishing the specific face matching the template from the two detected faces after the template applied was rotated 90 degrees. The reason resulted in this unsatisfying result was not because of the failure in the template matching algorithm nor the facial recognition processes, however, it was because the template was rotated 90 degrees resulted in different similarities returned which made the previous set threshold value invalid to distinguish different template matching results. Hence, the functionality of template matching algorithm largely depended on the threshold value set, in which case, the threshold value needed to be modified each time when the template or the video source was changed which made its application difficult because it was impossible to change the threshold values for different situations.

Thus, generally the application restriction of template matching algorithm was that it was not stable enough to match different objects with their templates without modifying the threshold value pre-set in order to make the algorithm work. More enhanced algorithms were to be integrated such as establishing databases to increase the stability of facial recognition to realize that even the templates or objects in the sources were zoomed or rotated, the programme was still able to recognize.

- **What were the factors that would affect the recognition accuracy?**

The accuracy of face detection and facial recognition were both covered in the recognition accuracy, therefore, all the factors influencing the accuracy of either one of them were the factors that would affect the recognition accuracy.

As discussed above, the template matching algorithm had its restrictions that the algorithm was likely to fail to realize its objective when either the template or the video source was modified owing to the different threshold values needed to be applied to make the algorithm fully operational. Besides the algorithm restrictions, the accuracy of face detection using CascadeClassifier and pretrained models would also be affected by certain factors which would affect the overall recognition accuracy.

It was discussed in the previous section “Face Detection results and discussions”, the different pretrained models applied in the CascadeClassifier generated different results. Neither of the two pretrained models applied was able to detect all the faces inside the given image without incorrectly highlighting the areas without faces. Each pretrained models had its pros and cons, and hence the one better finished the job was selected in this project but there was still possibility that it would fail.

Apart from the pretrained models applied, various factors would affect the recognition accuracy including the head rotation, tilt, lighting intensity, lighting angle, facial expression, scale (distance from the camera), aging, environmental noises and so on. Luckily in this project, these factors had not affected the overall facial recognition accuracy except for the head rotation. As shown in Figure 22, there was no face detected when the head was turned right about 60 degrees.

Figure 22. Capture from the Processed Video

The computer was not able to understand as human beings, therefore, though there was a face inside the frame, the programme was unable to recognize it because of the head rotation which caused the low similarity comparing the database of pretrained models. However, this project's objective was to enable the programme to recognize the faces looking directly towards the front only, therefore, this unperfect detection result was not affecting the overall facial recognition results. And this situation was only occurred once so that the overall detection accuracy was maintained at a high level. A proper message was then displayed indicating that no face was detected.

Lastly, another factor that would affect the recognition accuracy was the size. Both the sizes of the video frames and the template included. At the very beginning of this project, the "scaleFactor" was applied to resize the video frames to boost the face detection. Though the accuracy was to be lowered theoretically due to the loss of pixels, the facial detection worked fine, however, when the template matching algorithm was integrated, the programme failed to identify different faces so that the resizing of video frame was then removed. Also, there were size limitations set in the template matching algorithm. If the size of the template was larger than the maximum detectable size or smaller than the minimum, the algorithm would break down, therefore, the resize function was used to zoom the template which lowered the accuracy inevitably.

- **How was the proper similarity found to be considered as a "match" to template?**

The proper similarity was found by running the template matching algorithm once and outputting the similarities of different faces detected matching the template for each video frame as shown in Figure 23. Then a threshold value was set in between of the approximate means of two faces' similarities. Luckily, the difference between two similarities of the two faces in the video frames was great enough to set the threshold value easily.

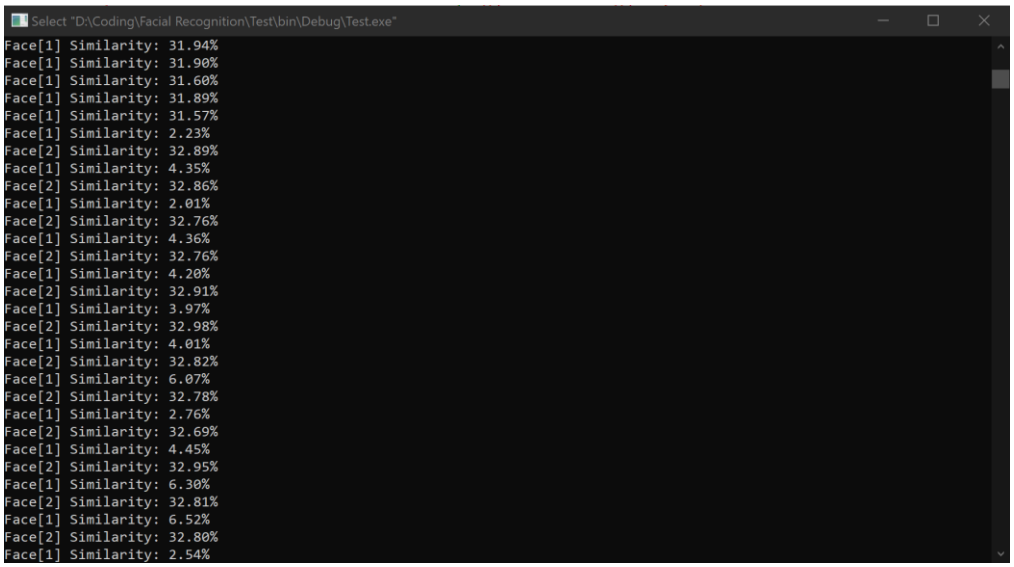

Figure 23. Template matching similarities

It was noticed from the similarities returned, the specific face of the researcher had similarities larger than 30.00% and another face detected had similarities lower than 10.00%, therefore, the threshold value was set in between which was set to 15.00%. Although the general similarities were relatively low due to certain setups in the template matching algorithm, the programme was still able to distinguish the different faces which had no effect on the overall facial recognition results.

Overall Project Results

The overall objective of this project was to apply facial recognition onto a digital video using template matching algorithm to realize face detection and identification during which the specific face matching the template was distinguished from other detected faces.

The algorithm developed in this project successfully realized the overall objectives. For the first 10 seconds, the programme was able to highlight the detected face perfectly knowing whom the face belonged to as shown in Figure 24. And for the next ten seconds, all the faces inside the video frames were detected and the specific face matching the template as shown in Figure 25 was identified as shown in Figure 26. Lastly, the processed video with the facial recognition done successfully was successfully generated after.

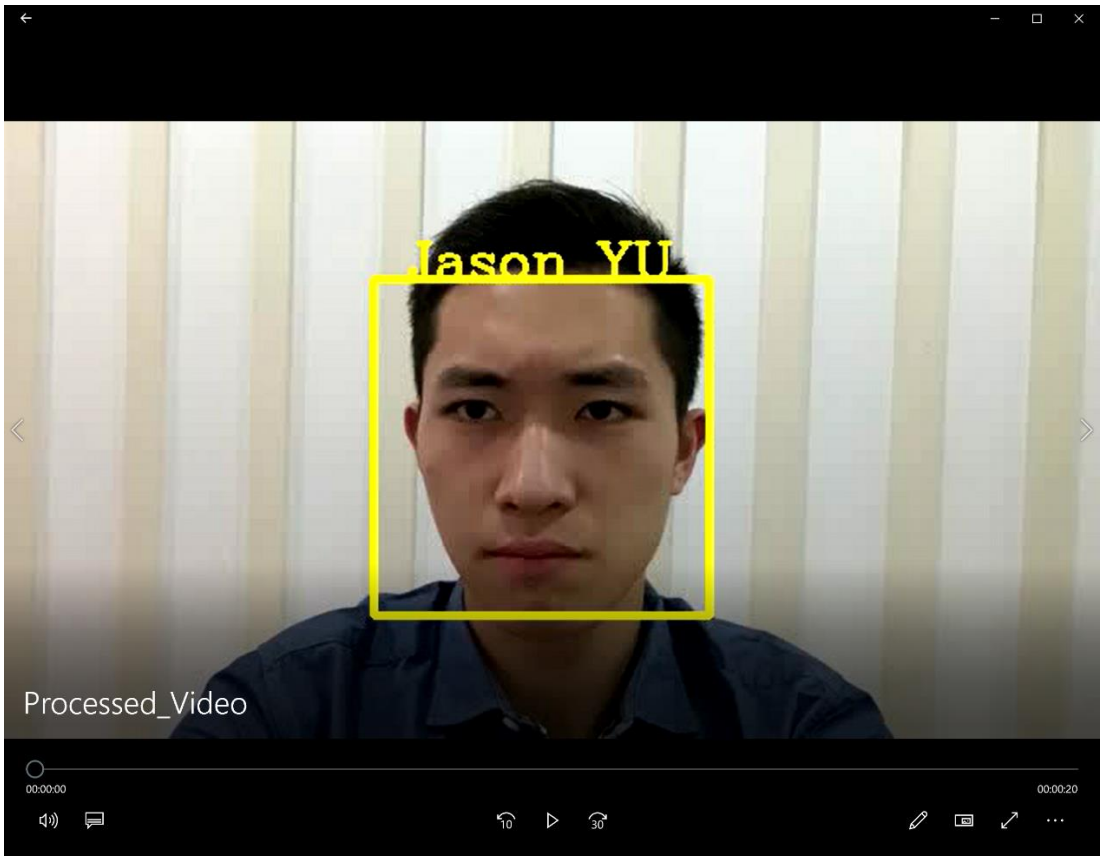

Figure 24. Overall facial recognition result for the first 10s

Figure 25. The Template

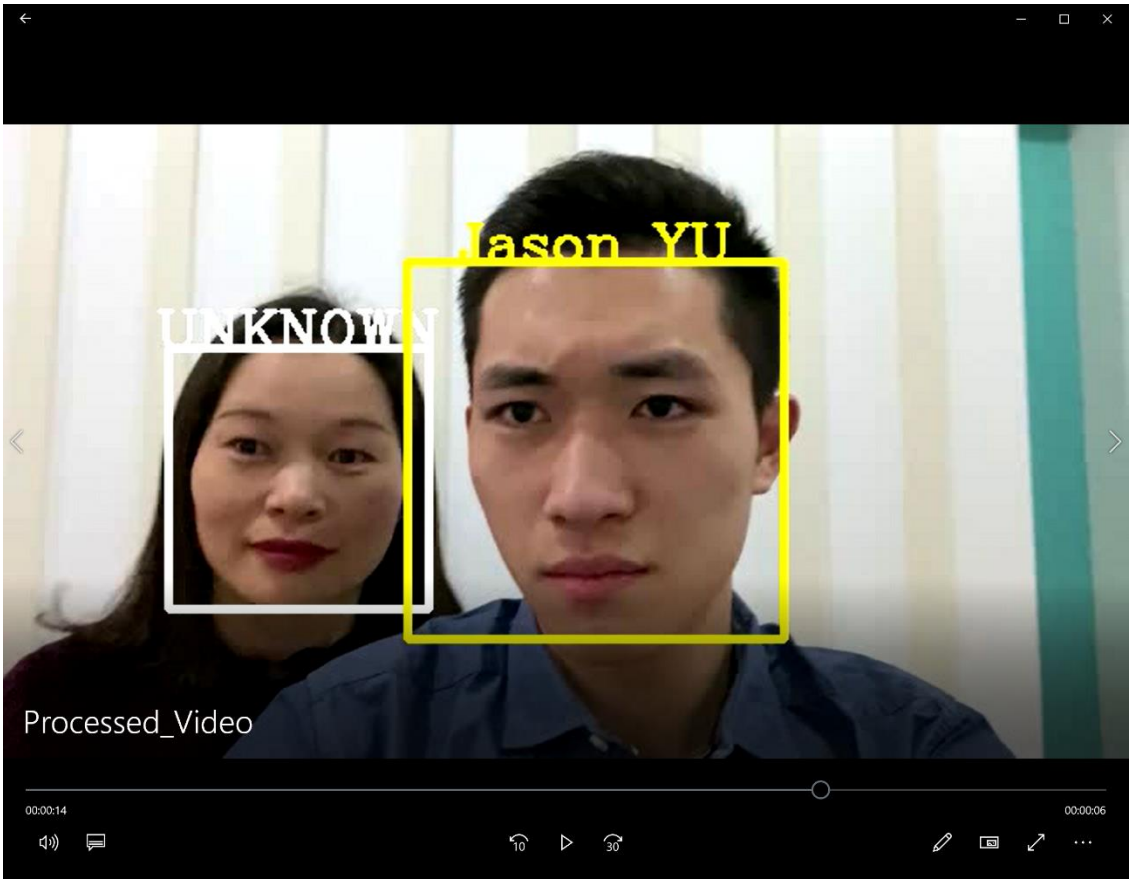

Figure 26. Overall facial recognition result for the second 10s

One advantage of this project was that the template used was not captured from the video frames. If the capture of the video frame was applied in the template matching algorithm, theoretically there would be one specific frame matching the template with a similarity of 100.00% which was impossible in real situations. Thus, if the programme was able to realize facial recognition using a third-party template, the programme was more advanced.

Conclusion

In this project, the algorithm for facial recognition in digital video using CodeBlocks integrated with OpenCV was studied. Various embedded classes and functions of OpenCV were tested and appropriately applied, different image processing methods were studied, different Haar feature-based pretrained models were tested and appropriately applied via CascadeClassifier to realize face detection and the template matching algorithm was studied and integrated in this facial recognition project. The pros and cons of the functions and algorithms applied in this project were discovered and discussed, application restrictions of the template matching algorithm and factors that would affect the recognition accuracy were discussed in detail.

The template matching algorithm was tested to be a very basic and fundamental facial recognition algorithm with low recognition accuracy and many application restrictions. In order to enhance the facial recognition programme, further re-deployment was needed.

In summary, the algorithm for facial recognition in digital video using CodeBlocks integrated with OpenCV was successfully developed which fully realized the objective of this project to detect and recognize faces in the digital video.

References

[1] https://opencv.org/about/ (no date) (Accessed: 21 April 2020).

[2] https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html (no date)

(Accessed: 21 April 2020).

[3]https://www.docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/template_matching/template_matching.html (no date) (Accessed: 21 April 2020).